UNIVERSITÉ PARIS-SACLAY 2023-2024 première session d'examen

Licence 3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique

vendredi 3 mai 2024 9h00 à 11h30 (durée : 2h30)

— *Aucun document n'est autorisé.*

— *Les programmes doivent être écrits en C/C++.*

— *Respecter les notations de l'énoncé : par exemple la variable notée n_t dans l'énoncé devra être écrite* `nt` *dans un programme, n'' devra être écrite* `ns`.

— *L'utilisation de fonctions est laissée à l'appréciation de l'étudiant, sauf dans les cas où elle est imposée.*

— *On suppose que tous les* `#include<iostream> ... #include<math.h> using namespace std;` *nécessaires sont sous-entendus, il n'y a pas à les écrire.*

— *Ne pas faire lire les données au clavier ou dans un fichier par un* `cin >>`*, ou l'équivalent pour un fichier, sauf si cela est explicitement demandé. Par défaut, les données seront donc fournies dans le programme lui-même, par des instructions du type :*

```
dx=0.01; a=1.7; b=1.1;
```

— *Tous les tableaux dont il s'agit dans l'énoncé sont des tableaux dynamiques, à déclarer avec un ou plusieurs* `malloc`.

— *Dans chaque exercice on suppose que le programme principal (s'il y en a un) et les fonctions (s'il y en a) sont écrits dans un unique fichier.*

— *Les 4 exercices sont indépendants les uns des autres.*

— *Le nombre de points (sur un total de 22) attribué à chaque question est indiqué entre parenthèses (c'est un barème indicatif). Un point est réservé à la qualité de la présentation, pour un total de 23 points.*

— *Le corrigé pourra être consulté sur le site du cours ultérieurement.*

1. Manipulation de coordonnées

On considère les coordonnées cartésiennes d'un ensemble de points dans un plan, représenté par le nombre de points et deux tableaux de valeurs en virgule flottante double précision.

```
int nb_points; double* x_points; double* y_points;
```

L'objectif de cet exercice est d'écrire quelques fonctions qui permettent de manipuler ces coordonnées. On suppose que les tableaux de coordonnées ont été alloués et les fonctions reçoivent en arguments les deux tableaux et le nombre de points. Vous ne devez allouer aucun tableau dans cet exercice.

Question 1 : (0.5 point)

Écrire une fonction C, ayant comme prototype :

```
void affiche_liste(int npt, double* x, double* y)
```

qui affiche le nombre de points, suivi des coordonnées (x, y) de tous les points.

Réponse à la question 1 :

```
void affiche_liste(int npt, double* x, double* y) {
  int i;
  cout << "N = " << npt << endl;
  for(i = 0; i < npt; i++)
    cout << x[i] << "," << y[i] << endl;
}
```

Question 2 : (0.5 point)

Écrire une fonction C, ayant comme prototype :

```
void change_origine(int npt, double* x, double* y, double xop, double yop)
```

qui effectue un changement d'origine des coordonnées, de $O \to O'$:

$$\begin{array}{rcl} O\,(0,0) & \to & O'\,(x_{O'}, y_{O'}) \\ x \to x' = x - x_{O'} & & y \to y' = y - y_{O'} \end{array}$$

Les nouvelles coordonnées seront stockées à la place des anciennes, dans les tableaux d'origine.

Réponse à la question 2 :

```
void change_origine(int npt, double* x, double* y, double xop, double yop) {
  int i;
  for(i = 0; i < npt; i++) {
    x[i] = x[i] - xop;
    y[i] = y[i] - yop;
  }
}
```

Question 3 : (1 point)

Écrire une fonction C, ayant comme prototype :

```
int le_plus_proche(int npt, double* x, double* y, double xp, double yp)
```

qui recherche et renvoie l'indice dans le tableau du point le plus proche, selon la distance euclidienne, du point de coordonnées (xp, yp).

Réponse à la question 3 :

```
int le_plus_proche(int npt, double* x, double* y, double xp, double yp) {
  double dx = xp - x[0];
  double dy = yp - y[0];
  double dc, dmin = dx*dx + dy*dy;
  int i, ip = 0;
  for(i = 1; i < npt; i++) {
    dx = xp - x[i];
    dy = yp - y[i];
    dc = dx*dx + dy*dy;
    if (dc < dmin) {
      ip = i;
      dmin = dc;
    }
  }
  return ip;
}
```

Question 4 : (1 point)

Écrire une fonction qui calcule les coordonnées suite à une rotation d'angle φ. Les nouvelles coordonnées remplaceront celles avant rotation, dans les tableaux d'origine. On rappelle :

$$\begin{aligned} x \to x' &= x\cos\varphi + y\sin\varphi \\ y \to y' &= -x\sin\varphi + y\cos\varphi \end{aligned}$$

Réponse à la question 4 :

```
void rotation(int npt, double* x, double* y, double phi) {
  double xc, yc, cphi = cos(phi), sphi = sin(phi);
  int i;
  for(i = 0; i < npt; i++) {
    // on doit copier la paire x[i], y[i], avant de les modifier par le calcul
    xc = x[i];
    yc = y[i];
    x[i] = xc*cphi + yc*sphi;
    y[i] = -xc*sphi + yc*cphi;
  }
}
```

Question 5 : (1 point)

Écrire une fonction qui lit les valeurs des coordonnées dans un fichier nommé *coordxy.txt* et les utilise pour remplir les tableaux `x` et `y`. Le fichier comprend au moins `npt` lignes, avec deux valeurs par ligne, séparées par un ou plusieurs espaces. Chaque paire de valeurs sur une ligne correpond aux coordonnées (x, y) d'un point.

Réponse à la question 5 :

```
void lit_fichier(int npt, double* x, double* y) {
  int i;
  fstream fich;
  fich.open("coordxy.txt", ios::in);
  for(i = 0; i < npt; i++)
    fich >> x[i] >> y[i];
  fich.close();
}
```

2. Pointeurs

Dans les programmes qui suivent les flèches `<---` signalent les lignes qui diffèrent entre les deux ou trois programmes de chaque question.
Pour chacun de ces programmes indiquer s'il est valide (c'est-à-dire si le programme s'exécutera sans message d'erreur), si non pourquoi, et si oui, le résultat affiché à l'écran.

Question 1 : (1 point)

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, *p;
  i = 3; *p = i;
  cout << i << " " << *p << endl;
  return 0;
}
```

Réponse :

Non valide :
`p` non initialisé par une adresse.

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, *p;
  i = 3; p = &i;
  cout << i << " " << *p << endl;
  return 0;
}
```

Réponse :

Valide :
3 3

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, *p;
  *p = 3; i = *p;          <------
  cout << i << " " << *p << endl;
  return 0;
}
```

Réponse :

Non valide :
`p` non initialisé par une adresse.

Question 2 : (1 point)

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int* p) {
  cout << *p << endl;
}
int main() {
  int i;
  i = 5;
  ff(i);
  return 0;
}
```

Réponse :

Non valide :
l'argument de `ff` est un pointeur qui doit recevoir une adresse et non un entier.

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int* p) {
  cout << *p << endl;
}
int main() {
  int i;
  i = 5;
  ff(&i);
  return 0;
}
```

Réponse :

Valide :
5

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int* p) {
  cout << p << endl;     <------
}
int main() {
  int i;
  i = 5;
  ff(&i);                <------
  return 0;
}
```

Réponse :

Valide :
0x7fff4d21f6dc, par exemple c'est l'adresse de `i` en hexadécimal.

Question 3 : (1 point)

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int i) {
  cout << i << endl;
}
int main() {
  int j, *p;
  p = &j;
  j = 5;
  ff(j);
  return 0;
}
```

Réponse :

Valide :
5
pointeur non utilisé.

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int i) {
  cout << i << endl;
}
int main() {
  int j, *p;
  p = &j;
  j = 5;
  ff(*p);
  return 0;
}
```

Réponse :

Valide :
5

```
#include<iostream>
#include<stdlib.h>
using namespace std;
void ff(int i) {
  cout << i << endl;
}
int main() {
  int j, *p;
  p = &j;
  j = 5;
  ff(p);              <------
  return 0;
}
```

Réponse :

Non valide :
l'argument de `ff` est un entier qui ne peut recevoir l'adresse contenue dans `p`.

Question 4 : (1 point)

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, n = 5;
  double* t = (double*)malloc(n*sizeof(double));

  for(i = 0; i < n; i++) t[i] = 1;
  for(i = 0; i < n; i++) cout << t[i] << " ";
  cout << endl;
  return 0;
}
```

Réponse :

Valide :
1 1 1 1 1

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main() {
  int i, n = 5;
  double* t = (double*)malloc(n*sizeof(double));
  double* p;                                    <------
  for(p = t; p < t+n; p++) *p = 1;              <------
  for(i = 0; i < n; i++) cout << t[i] << " ";
  cout << endl;
  return 0;
}
```

Réponse :

Valide :
1 1 1 1 1
on peut ajouter, retrancher un entier à un pointeur et faire des comparaisons entre pointeurs du même type.

3. Trajectoire de Mercure

En relativité générale, l'équation qui décrit le mouvement de la planète Mercure sous l'influence du Soleil dans le plan équatorial du Soleil est

$$\frac{d^2u}{d\varphi^2} + u = \frac{r_s^2 c^2}{2\ell^2} + \frac{3}{2}u^2 \tag{1}$$

Ici $u(\varphi) = r_s/r(\varphi)$ avec $r_s = 2GM/c^2$ le rayon de Schwarzschild du Soleil ($M = 1.988 \cdot 10^{30}$ kg est la masse du Soleil et bien sûr $G = 6.674 \cdot 10^{-11}$ $\mathrm{m^3\,kg^{-1}\,s^{-2}}$ et $c = 2.998 \cdot 10^8$ $\mathrm{m\,s^{-1}}$) et ℓ est le moment cinétique par unité de masse de Mercure, qui est conservé. On décrit donc la trajectoire en coordonnées polaires (r, φ) dans le plan sous la forme d'une fonction $r(\varphi) = r_s/u(\varphi)$ où $u(\varphi)$ satisfait à l'équation différentielle de second ordre ci-dessus. Noter qu'on ne cherche pas à déterminer la position au cours du temps de Mercure. En mécanique classique on aurait la même équation différentielle mais sans le terme non linéaire $\frac{3}{2}u^2$.

On donne également la relation

$$\ell^2 = \frac{a r_s c^2 (1 - \epsilon^2)}{2} \tag{2}$$

avec $a = 5.791 \cdot 10^{10}$ m le demi-grand axe de l'orbite quasi-elliptique de Mercure et $\epsilon = 0.2056$ son excentricité.

Question 1 : (0.5 point)

Déclarer et initialiser les constantes du problème (`M`, `G`, `c`, `rs`, `a`, `eps`, `l2`) comme variables globales.

Réponse à la question 1 :

```
double M = 1.988e30, G = 6.674e-11, c = 2.998e8;
double rs = 2*G*M/c/c;
double a = 5.791e10, eps = 0.2056;
double l2 = a*rs*c*c*(1-eps*eps)/2;
```

Question 2 : (1 point)

Écrire une fonction avec le prototype `void systeme(double* q, double phi, double* qp, int n)` qui décrit l'équation différentielle (1) pour u dans une forme qui peut être utilisée par la méthode d'Euler ou la méthode RK4.

Réponse à la question 2 :

```
void systeme(double* q, double phi, double* qp, int n){
  qp[0] = q[1];
  qp[1] = -q[0] + rs*rs*c*c/2/l2 + 3./2.*q[0]*q[0];
}
```

Question 3 : (1 point)

Écrire une fonction avec le prototype :

```
void euler(void (*syst)(double*,double,double*,int), double* q, double phi, double dphi, int n)
```

qui fait une itération de la méthode d'Euler.

Réponse à la question 3 :

```
void euler(void (*syst)(double*,double,double*,int),
           double* q, double phi, double dphi, int n){
  int i;
  double* qp = (double*)malloc(n * sizeof(double));
  syst(q, phi, qp, n);
  for (i = 0; i < n; i++)
    q[i] = q[i] + dphi * qp[i];
  free(qp);
}
```

Question 4 : (1.5 points)

Écrire le programme principal (fonction `int main()`) qui calcule la variable u en $N = 10^7$ valeurs de φ allant de $\varphi = 0$ à $\varphi = 20\,000\pi$ rad en utilisant la méthode d'Euler, avec comme conditions initiales $u(\varphi = 0) = r_s^2 c^2 (1 + \epsilon)/(2\ell^2)$ et $du/d\varphi(\varphi = 0) = 0$. Le programme doit écrire toutes les N valeurs de φ, $x = r\cos\varphi$ et $y = r\sin\varphi$ (donc trois valeurs par ligne) dans un fichier nommé *resultats.res*.

Réponse à la question 4 :

```
int main(){
  int i, n = 2, N = 1e7;
  double phi = 0, phi_fin = 20000*M_PI, dphi = phi_fin/(N-1);
  double r, x, y;
  double* q = (double*)malloc(n * sizeof(double));
  fstream fich;
                                    // <-- Endroit (A) pour ajouts question 5
  q[0] = rs*rs*c*c*(1+eps)/2/l2;
  q[1] = 0;
                                    // <-- Endroit (B) pour ajouts question 5
  fich.open("resultats.res", ios::out);
  for(i = 0; i < N; i++){
    r = rs/q[0];
    x = r*cos(phi); y = r*sin(phi);
                                    // <-- Endroit (C) pour ajouts question 5
    fich << phi << " " << x << " " << y << endl;
    euler(systeme, q, phi, dphi, n);
    phi += dphi;
  }
  fich.close();
  free(q);
  return 0;
}
```

A noter qu'avec les paramètres donnés, le programme ne fonctionne pas correctement avec la méthode d'Euler, il faudrait utiliser RK4.

Question 5 : (1 point)

Rajouter au programme la fonction et les lignes nécessaires pour calculer à chaque valeur de φ la distance entre Mercure selon la trajectoire relativiste (calculée dans la question précédente) et Mercure selon la trajectoire classique, qui est la solution de (1) sans le terme $\frac{3}{2}u^2$ (avec les mêmes conditions initiales). Faire écrire cette distance comme quatrième colonne dans le fichier *resultats.res*.

Réponse à la question 5 :

D'abord on ajoute la fonction suivante :

```
void systeme_cl(double* q, double phi, double* qp, int n){
  qp[0] = q[1];
  qp[1] = -q[0] + rs*rs*c*c/2/l2;
}
```

Puis on ajoute à l'endroit (A) indiqué dans la réponse à la question précédente :

```
  double r_cl, dist;
  double* q_cl = (double*)malloc(n * sizeof(double));
```

et à l'endroit (B) :

```
  q_cl[0] = q[0];
  q_cl[1] = q[1];
```

et à l'endroit (C) :

```
    r_cl = rs/q_cl[0];
    dist = fabs(r - r_cl); // on compare deux points avec la meme valeur de phi
```

Ensuite on change la ligne suivante en

```
    fich << phi << " " << x << " " << y << " " << dist << endl;
```

et on ajoute tout de suite après

```
    euler(systeme_cl, q_cl, phi, dphi, n);
```

Enfin on ajoute `free(q_cl);` avant le `return 0;`.

4. Emittance d'un faisceau de particules

L'emittance ϵ d'un faisceau est proportionnelle à l'aire occupée par les particules dans l'espace des phases (x, x') du faisceau. Lorsqu'une ellipse est tracée autour de la distribution des particules dans l'espace des phases, l'équation de l'ellipse est la suivante :

$$\gamma x^2 + 2\alpha x x' + \beta x'^2 = \delta \tag{3}$$

où δ est proportionnel à l'aire de l'ellipse et est donc une borne supérieure de ϵ.

A noter que l'allocation de tous les tableaux sera faite dans la fonction `main`, à écrire dans la dernière question.

Question 1 : (1 point)

Ecrire une fonction `distri` prenant en argument deux tableaux à un indice `x` et `xp` et un entier `N`. La fonction doit générer les N composantes aléatoires des tableaux `x` et `xp`. Les coordonnées (x, x') ainsi générées doivent être distribuées uniformément dans une ellipse centrée en $(0, 0)$ ayant pour paramètres $\delta = 5.0$, $\beta = 20.0$, $\alpha = -5.0$, $\gamma = (1 + \alpha^2)/\beta$ et de demi-grand axe $a = 10.3$. Pour faire cela, tirer des coordonnées distribuées uniformément dans un carré de côté $2a$ centré en $(0, 0)$ et écarter les points en dehors de l'ellipse (c'est-à-dire les points pour lesquels le côté gauche de l'équation (3) est supérieur à δ). Utiliser la fonction `drand48()` de la bibliothèque `stdlib.h`.

Réponse à la question 1 :

```
void distri(double* x, double* xp, int N){
  double a = 10.3, beta = 20.0, alpha = -5.0, delta = 5.0;
  double gamma = (1 + alpha*alpha)/beta;
  int k;
  for(k = 0; k < N; k++)
    do{
      x[k] = 2*a*drand48() - a;
      xp[k] = 2*a*drand48() - a;
    }while(gamma*x[k]*x[k] + 2*alpha*x[k]*xp[k] + beta*xp[k]*xp[k] > delta);
}
```

Nous allons ensuite nous intéresser à l'émittance de la distribution simulée. L'émittance est définie telle que :

$$\epsilon = \sqrt{\sigma_x^2 \sigma_{x'}^2 - \sigma_{xx'}^2} \tag{4}$$

avec σ_x et $\sigma_{x'}$ les écarts-types, dont on rappelle la définition $\sigma_x = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$, et

$$\sigma_{xx'} = \frac{\sum_i (x_i - \langle x \rangle)(x'_i - \langle x' \rangle)}{N} \tag{5}$$

Question 2 : (1.75 points)

1. Ecrire une fonction avec prototype `double var_et_moy(double* x, int N, double* moy)` qui calcule et renvoie la variance ($= \sigma_x^2$) et la moyenne des N points du tableau `x` (la moyenne est renvoyée en utilisant le pointeur `moy`).
2. Ecrire une fonction qui calcule et renvoie la variance croisée $\sigma_{xx'}$.
3. Ecrire une fonction `emittance` qui prend en argument les deux tableaux `x` et `xp` et leur taille `N` et qui retourne l'émittance ϵ de la distribution de particules.

Réponse à la question 2 :

```
double var_et_moy(double* x, int N, double* moy){
  int i;
  double sum2 = 0.;
  *moy = 0.;
  for(i = 0; i < N; i++){
    *moy += x[i];
    sum2 += x[i]*x[i];
  }
  *moy /= N;
  return sum2/N - (*moy)*(*moy);
}
```

```
double var_croisee(double* x, double* xp, int N, double moy_x, double moy_xp){
  int i;
  double sig_xxp = 0.;
  for(i = 0; i < N; i++)
    sig_xxp += (x[i]-moy_x) * (xp[i]-moy_xp);
  return sig_xxp/N;
}

double emittance(double* x, double* xp, int N){
  double moy_x, moy_xp;
  double sig_x2, sig_xp2, sig_xxp;
  sig_x2 = var_et_moy(x, N, &moy_x);
  sig_xp2 = var_et_moy(xp, N, &moy_xp);
  sig_xxp = var_croisee(x, xp, N, moy_x, moy_xp);
  return sqrt(sig_x2*sig_xp2 - sig_xxp*sig_xxp);
}
```

Question 3 : (1.75 points)

Ecrire une fonction `hist2D` qui va créer un histogramme 2D de la distribution de particules précédente, c'est-à-dire énumérer les nombres de particules dont les coordonnées tombent dans de petits intervalles de x et de x'. Pour cela, on coupe l'intervalle $[-10, 10]$ pour x en `NbXpoint`=40 petits intervalles de taille `DeltaX`= 0.5, qu'on dénote par la valeur au centre de chaque intervalle. De la même façon, on coupe l'intervalle $[-3, 3]$ pour x' en `NbXppoint`=60 petits intervalles de taille `DeltaXp`= 0.1. Déclarer en variables globales `XMin, XMax, XpMin, XpMax, DeltaX, DeltaXp, NbXpoint` et `NbXppoint`.

La fonction `hist2D` prendra en argument les tableaux `x` et `xp` et leur taille `N`, ainsi qu'un tableau à deux indices `double** hist` de `NbXpoint`×`NbXppoint` lignes et 3 colonnes. La fonction doit remplir le tableau `hist` de telle façon que la première colonne contienne les valeurs des centres des petits intervalles de x, la deuxième colonne contienne les valeurs des centres des petits intervalles de x' et la dernière colonne contienne le nombre de particules dont les coordonnées tombent dans ces deux petits intervalles de x et de x'. [Explicitement, les premières 60 lignes auront donc la même valeur -9.75 dans la première colonne tandis que la deuxième colonne parcourt toutes ses valeurs possibles, ensuite les 60 lignes suivantes auront la valeur -9.25 dans la première colonne, etc.]

On se référera dans la suite de l'exercice à chaque ligne du tableau comme un canal de l'histogramme.

Réponse à la question 3 :

```
double XMin = -10.0; double XMax = -XMin; double DeltaX = 0.5;
double XpMin = -3.0; double XpMax = -XpMin; double DeltaXp = 0.1;
int NbXpoint = (XMax-XMin)/DeltaX;
int NbXppoint = (XpMax-XpMin)/DeltaXp;

void hist2D(double* x, double* xp, int N, double** hist){
  int i, j, l, k = 0, compteur;
  double fX = XMin + DeltaX/2, fXp;
  for(i = 0; i < NbXpoint; i++){
    fXp = XpMin + DeltaXp/2;
    for(j = 0; j < NbXppoint; j++){
      hist[k][0] = fX;
      hist[k][1] = fXp;
      compteur = 0;
      for(l = 0; l < N; l++)
        if(x[l] >= fX - DeltaX/2 && x[l] < fX + DeltaX/2
           && xp[l] >= fXp - DeltaXp/2 && xp[l] < fXp + DeltaXp/2)
          compteur++;
      hist[k][2] = compteur;
      k++;
      fXp += DeltaXp;
    }
    fX += DeltaX;
  }
}
```

A noter que pour remplir un histogramme avec des intervalles de taille fixe comme ici, il serait en fait plus efficace de faire une boucle seulement sur les `N` points des tableaux `x` et `xp` (et non pas trois boucles imbriquées) et de déterminer pour chaque point quel compteur (pour quel intervalle) de l'histogramme il faut augmenter.

Dans la suite de l'exercice nous allons simuler une mesure d'émittance à partir de la distribution de particules générée en question 1, avec un bruit gaussien ajouté aux canaux pour simuler de vraies mesures. Le but est de comparer une méthode d'analyse d'émittance avec la valeur connue de départ (calculée en question 2).

On suppose qu'une fonction de prototype `double emittance_mes(double** hist)` existe (il n'est pas demandé de la programmer) qui calcule l'émittance "mesurée" à partir d'un histogramme comme créé dans la question précédente. Vous pouvez faire appel à cette fonction librement.

Question 4 : (0.25 point)

Ecrire une fonction `normale` prenant en argument `double sigma` et `double mu` et qui retourne une variable aléatoire suivant une loi normale de centre μ et d'écart-type σ. Pour cela, utiliser la transformation de Box-Muller qui prend deux variables aléatoires u_1 et u_2 uniformes sur l'intervalle $[0, 1]$ et crée la variable aléatoire gaussienne $z = \sigma\sqrt{-2\ln(u_1)}\cos(2\pi u_2) + \mu$.

Réponse à la question 4 :

```
double normale(double sigma, double mu){
  return sigma * sqrt(-2.*log(drand48())) * cos(2*M_PI*drand48()) + mu;
}
```

Question 5 : (1 point)

Ecrire une fonction `threshold` prenant en argument le tableau `hist` et une variable `pourcent`. Cette fonction doit retirer à chaque canal de l'histogramme `hist` la même valeur seuil qui vaut un certain pourcentage de la valeur du canal maximum. Si le résultat de cette soustraction est négatif, il faut mettre zéro dans le canal.

Réponse à la question 5 :

```
void threshold(double** hist, double pourcent){
  int k, Nbline = NbXpoint*NbXppoint;
  double max = 0, seuil;
  for(k = 0; k < Nbline; k++)
    if(hist[k][2] > max)
      max = hist[k][2];
  seuil = max*pourcent/100;
  for(k = 0; k < Nbline; k++){
    hist[k][2] -= seuil;
    if (hist[k][2] < 0)
      hist[k][2] = 0;
  }
}
```

Question 6 : (1 point)

Ecrire une fonction `droite` qui prend en argument deux tableaux à un indice `double* x` et `double* y` ainsi que la taille (commune) des tableaux. Celle-ci doit retourner le paramètre d de l'équation linéaire $y = cx + d$. Les valeurs optimales d'ajustement des paramètres de la droite selon le critère des moindres carrés sont données par les relations :

$$\begin{cases} c = \sigma_{xy}/\sigma_x^2 \\ d = \langle y \rangle - c\langle x \rangle \end{cases} \tag{6}$$

Réponse à la question 6 :

```
double droite(double* x, double* y, int Nbpoint){
  double moy_x, moy_y, sigx2, cov;
  double c, d;
  sigx2 = var_et_moy(x, Nbpoint, &moy_x);
  c = var_et_moy(y, Nbpoint, &moy_y);
  cov = var_croisee(x, y, Nbpoint, moy_x, moy_y);
  c = cov/sigx2;
  d = moy_y - c*moy_x;
  return d;
}
```

Question 7 : (2.25 points)

Ecrire la fonction `main`. Celle-ci doit :

1. Allouer tous les tableaux et les libérer à la fin.
2. Calculer l'émittance d'une distribution de $N = 10000$ particules.
3. Créer l'histogramme de cette distribution.
4. Ajouter un bruit gaussien ($\mu = 12.0$, $\sigma = 3.0$) à chaque canal.
5. Retirer progressivement le bruit et calculer à chaque itération l'émittance à partir de l'histogramme. Pour cela prendre comme pourcentages du seuil $35, 36, \ldots, 75$ %. N'oubliez pas de d'abord créer une copie de l'histogramme, afin que vous puissiez repartir de l'histogramme bruité original pour le calcul de chaque seuil.
6. Faire un fit linéaire sur les valeurs d'emittance obtenues.
7. Afficher à l'écran l'émittance calculée en point 2. et l'émittance extrapolée en seuil = 0 % à l'aide du fit linéaire.

Bien sûr, la fonction `main` doit également contenir toutes les autres commandes nécessaires pour créer un programme qui fonctionne correctement.

Réponse à la question 7 :

```
int main(){
  int i, j, k, N = 10000, Nbline = NbXpoint*NbXppoint;
  int N_fin = 75, N_start = 35;
  int Nbpoint = N_fin - N_start + 1;
  double e, d, sigma = 3.0, mu = 12.0;
  double* x = (double*)malloc(N * sizeof(double));
  double* xp = (double*)malloc(N * sizeof(double));
  double **hist, **hist_copie;
  double* seuil = (double*)malloc(Nbpoint * sizeof(double));
  double* emit_calc = (double*)malloc(Nbpoint * sizeof(double));

  srand48(time(NULL));
  hist = (double**)malloc(Nbline * sizeof(double*));
  hist_copie = (double**)malloc(Nbline * sizeof(double*));
  for (i = 0; i < Nbline; i++){
    hist[i] = (double*)malloc(3 * sizeof(double));
    hist_copie[i] = (double*)malloc(3 * sizeof(double));
  }
  distri(x, xp, N);
  e = emittance(x, xp, N);
  hist2D(x, xp, N, hist);
  for(i = 0; i < Nbline; i++){
    hist[i][2] += normale(sigma, mu);
    for(j = 0; j < 3; j++)
      hist_copie[i][j] = hist[i][j];
  }
  for(k = 0; k < Nbpoint; k++){
    seuil[k] = k + N_start;
    threshold(hist, seuil[k]);
    emit_calc[k] = emittance_mes(hist);
    for(i = 0; i < Nbline; i++)
      for(j = 0; j < 3; j++)
        hist[i][j] = hist_copie[i][j];
  }
  d = droite(seuil, emit_calc, Nbpoint);
  cout << "L'emittance calculee du faisceau est : " << e << endl;
  cout << "L'emittance reconstruite est : " << d << endl;

  free(x); free(xp);
  free(seuil);
  free(emit_calc);
  for(i = 0; i < Nbline; i++){
```

```
    free(hist[i]);
    free(hist_copie[i]);
  }
  free(hist); free(hist_copie);
  return 0;
}
```